

(19)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11)

EP 0 855 648 A2

(12)

EUROPEAN PATENT APPLICATION

(43) Date of publication:
29.07.1998 Bulletin 1998/31

(51) Int Cl. 6: G06F 9/38

(21) Application number: 98300494.6

(22) Date of filing: 23.01.1998

(84) Designated Contracting States:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC
NL PT SE

Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 24.01.1997 US 36223 P

(71) Applicant: TEXAS INSTRUMENTS INC.
Dallas, Texas 75243 (US)

(72) Inventors:

- Simar, Laurence R.
Richmond Texas 77469 (US)

- Seshan, Natarajan
Houston Texas 77063-1234 (US)
- Tatge, Reid E.
Richmond Texas 77469 (US)
- Davis, Alan L.
Sugarland Texas 77477 (US)

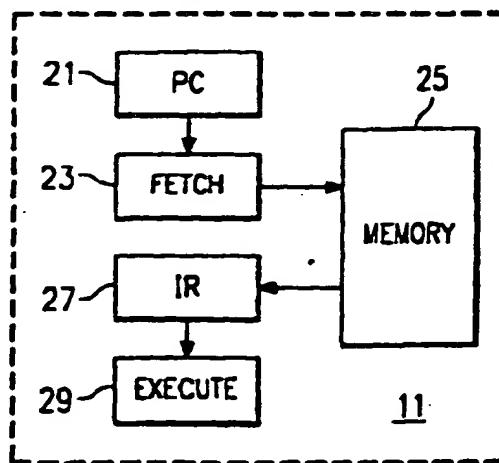
(74) Representative: Potter, Julian Mark et al
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Data processing with parallel or sequential execution of program instructions

(57) A data processing system 10 includes data processing circuitry 11 having circuitry (21, 23 and 25) for producing a set of instructions which include respective instruction portions for indicating whether the respective instructions can be executed simultaneously

with another of the instructions. Program execution circuitry (29) receives the set of instructions and is selectively responsive to the instructive portions for executing simultaneously a plurality of the instructions indicated by the instruction portions.

FIG. 2



Description

This invention relates generally to data processing and, more particularly but not exclusively, to data processing with both parallel and sequential execution of program instructions.

5 Data processing systems and data processors are used in myriad applications which in turn have an impact on virtually every aspect of life. The utility of these myriad applications can ordinarily be enhanced by increasing the speed and throughput of the associated data processing systems and data processors.

10 One way to enhance speed and throughput is, where possible, to execute program instructions in parallel rather than in sequential fashion. One known approach in this regard is to utilize a special mode instruction which specifies parallel or sequential execution of program instruction. Another known approach in this regard is to use a mask to specify null instructions in a parallel type packet. Another known solution is to perform data processing in a parallel mode only.

15 Although the aforementioned techniques are capable of improving speed and throughput, they are nevertheless undesirably difficult to implement and disadvantageously costly in terms of processing overhead.

It is therefore desirable to provide for parallel execution of program instructions in a manner which reduces the implementational difficulties and processing overhead associated with the above-described approaches.

An object of a preferred embodiment of the present invention is to utilize a portion of a given program instruction to determine whether that instruction can be executed simultaneously with another program instruction.

20 In general, and in a form of the present invention, a data processing device is provided which has circuitry for producing a set of instructions which include instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions. The data processing circuitry includes program execution circuitry connected to the producing circuitry for receiving the set of instructions and selectively executing simultaneously a plurality of the instructions in response to indicator portions.

25 In another form of the present invention, a preferred embodiment comprises a method for operating a central processing unit (CPU) within a data processing device comprises the steps of: providing a set of instructions with respective instruction portions for indicating whether the respective instructions can be executed simultaneously; and determining from the instruction portions whether a plurality of the instructions can be executed simultaneously.

Other embodiments of the present invention will be evident from the description and drawings.

30 Embodiments in accordance with the present invention will now be further described by way of example, with reference to the accompanying drawings in which:

Fig. 1 is a block diagram of a data processing system according to the present invention;

Fig. 2 is a block diagram of a portion of the data processing circuitry of Fig. 1;

Fig. 3 illustrates the basic format of an instruction packet utilized in the present invention;

35 Fig. 4 is one example of an instruction packet according to the format of Fig. 3;

Fig. 5 illustrates the execution sequence defined by the instruction packet of Fig. 4;

Fig. 6 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 7 illustrates the execution sequence defined by the instruction packet of Fig. 6;

Fig. 8 illustrates another example of an instruction packet according to the format of Fig. 3;

40 Fig. 9 illustrates the execution sequence defined by the instruction packet of Fig. 8;

Fig. 10 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 11 illustrates the execution sequence defined by the instruction packet of Fig. 10;

Fig. 12 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 13 illustrates the execution sequence defined by the instruction packet of Fig. 12;

45 Fig. 14 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 15 illustrates the execution sequence defined by the instruction packet of Fig. 14;

Fig. 16 illustrates another example of an instruction packet according to the format of Fig. 3;

Fig. 17 illustrates the execution sequence defined by the instruction packet of Fig. 16;

Fig. 18 illustrates another example an instruction packet according to the format of Fig. 3;

50 Fig. 19 illustrates the execution sequence defined by the instruction packet of Fig. 18;

Figure 20 is a block diagram of a microprocessor which has an embodiment of the present invention;

Figure 21 is a block diagram of the execution units and register files of the microprocessor of Fig. 20;

Figure 22A is a chart which illustrates the processing phases of an instruction execution pipeline in the microprocessor of Fig. 20;

55 Figure 22B is a chart which illustrates the execution phases of the instruction execution pipeline in the microprocessor of Fig 20;

Figure 23 is a timing diagram which illustrates timing details of processing an instruction fetch packet during the processing phases of Fig. 22A and execution of the execution packet during the execution phases of Fig. 22B;

Figure 24 is a block diagram showing instruction dispatching in the microprocessor of Fig. 20;
 Figure 25 illustrates the basic format of an instruction fetch packet for the microprocessor of Fig. 20; and
 Figure 26 illustrates an 8-word fetch packet that is partially parallel.

5 Corresponding numerals and symbols in the different figures and tables refer to corresponding parts unless otherwise indicated.

Fig. 1 is a block diagram of a data processing system 10 according to an exemplary embodiment of the present invention. The data processing system 10 includes data processing circuitry 11 and peripheral circuitries 13, 15, 17 and 19. In the exemplary embodiment of Fig. 1, the data processing circuitry 11 is connected to each of the peripheral circuitries 13, 15, 17 and 19 for transfer of information between data processing circuitry 11 and peripheral circuitries 13, 15, 17 and 19. However, and as will be apparent from the following description, a data processing system according to an embodiment of the present invention could include any quantity and type of peripheral circuitries and peripheral devices (such as peripherals 13, 15, 17 and 19) interconnected among themselves and with data processing circuitry 11 in any manner heretofore or hereafter conceivable to workers in the art.

10 Fig. 2 illustrates a portion of one exemplary embodiment of the data processing circuitry 11 of Fig. 1. In Fig. 2, fetch circuitry 23 accesses memory 25 at an address specified by program counter 21 and causes an instruction packet at that address to be loaded into instruction register 27. Program execution circuitry 29 decodes and executes the instructions of the instruction packet held in instruction register 27.

15 Fig. 3 illustrates the basic format of the instruction packet fetched from memory 25. In the disclosed exemplary embodiment, an instruction packet includes four 32-bit instructions A, B, C and D. As shown in Fig. 3, instructions A, B, C and D are stored at consecutive addresses in memory 25. Thus, during normal sequential execution of program instructions, instruction A would be executed first, followed sequentially by instructions B, C and D.

20 Bit 0 of each instruction in Fig. 3 has been designated as a p-bit. The p-bits define how the instructions will be executed. The p-bits of the Fig. 3 instruction packet are inspected from left to right by the program execution circuitry 29. If the p-bit of a given instruction is equal to logic one, then the next sequential instruction in the packet is to be executed in parallel with the first-mentioned instruction. Program execution circuitry 29 applies this rule until an instruction in the instruction packet is reached with a p-bit equal to logic 0.

25 If a given instruction has a p-bit of 0, then the next sequential instruction is executed sequentially after the given instruction (and after any instructions which are executed in parallel with the given instruction). The program execution circuitry 29 applies this rule until it reaches an instruction in the instruction packet with a p-bit of logic 1.

30 Figs. 4-19 provide application examples of the above-described p-bit rules.

35 Fig. 4 illustrates an instruction packet in which all p-bits are 0. Thus, instructions A-D are executed sequentially as shown in Fig. 5.

40 Fig. 6 illustrates an instruction packet in which the p-bits of instructions A, B and C are equal to 1, and the p-bit of instruction D is 0. Thus, instructions A, B, C and D are executed simultaneously, that is, in parallel as shown in Fig. 7.

45 In the instruction of Fig. 8, only the p-bit of instruction C is set to one, resulting in the execution sequence of Fig. 9, namely, instructions A and B are executed sequentially, followed by instructions C and D which are executed in parallel.

50 In the instruction packet of Fig. 10, only the p-bit of instruction B is set to one, resulting in the execution sequence shown in Fig. 11, namely instruction A is executed and then followed sequentially by the parallel execution of instructions B and C, which is then followed sequentially by execution of instruction D.

55 In the instruction packet of Fig. 12, the p-bits of instructions B and C are set to one, and the p-bits of instructions A and D are zero. This results in the instruction sequence shown in Fig. 13, namely instruction A is executed and is then sequentially followed by the parallel execution of instructions B, C and D.

60 In the instruction packet of Fig. 14, only the p-bit of instruction A is set to logic one, resulting in the execution sequence shown in Fig. 15, namely instructions A and B are executed in parallel and then followed sequentially by the execution of instruction C and then the execution of instruction D.

65 In the instruction packet of Fig. 16, the p-bits of instructions A and C are set to one and the p-bits of instructions B and D are 0, resulting in the execution sequence illustrated in Fig. 17, namely the parallel execution of instructions A and B followed sequentially by the parallel execution of instructions C and D.

70 In the instruction packet of Fig. 18, the p-bits of instructions A and B are set to 1 and the p-bits of instructions C and D are 0. This results in the execution sequence illustrated in Fig. 19, namely instructions A, B and C are executed in parallel and then followed sequentially by execution of instruction D.

75 Because the instruction packet in the disclosed example includes 4 program instructions, the program compiler can always provide instruction D (the fourth instruction) with a p-bit of 0. The compiler determines the values of the remaining p-bits of instructions A, B and C based on the propriety of executing instructions A and B in parallel, the propriety of executing instructions B and C in parallel, and the propriety of executing instructions A, B and C in parallel. For example, if execution of instruction B requires a result provided by execution of instruction A, then the compiler

would provide instruction A with a p-bit of 0 so that instruction B would be executed sequentially after instruction A. As another example, if instructions B and C access the same register, then the compiler would provide instruction B with a p-bit of 0 to ensure that instructions B and C are executed sequentially rather than in parallel.

Figure 20 is a block diagram of a microprocessor 1 which has an embodiment of the present invention. Microprocessor 1 is a VLIW digital signal processor ("DSP"). In the interest of clarity, Figure 20 only shows those portions of microprocessor 1 that are relevant to an understanding of an embodiment of the present invention. Details of general construction for DSPs are well known, and may be found readily elsewhere. For example, U.S. Patent 5,072,418 issued to Frederick Boutaud, et al, describes a DSP in detail and is incorporated herein by reference. U.S. Patent 5,329,471 issued to Gary Swoboda, et al, describes in detail how to test and emulate a DSP and is incorporated herein by reference. Details of portions of microprocessor 1 relevant to an embodiment of the present invention are explained in sufficient detail hereinbelow, so as to enable one of ordinary skill in the microprocessor art to make and use the invention.

In microprocessor 1 there are shown a central processing unit (CPU) 10, data memory 22, program memory 23, peripherals 60 and an external memory interface (EMIF) with a direct memory access (DMA) 61. CPU 10 further has an instruction fetch/decode unit 10a-c, a plurality of execution units, including an arithmetic and load/store unit D1, a multiplier M1, an ALU/shifter unit S1, an arithmetic logic unit ("ALU") L1, a shared multiport register file 20a from which data are read and to which data are written. Decoded instructions are provided from the instruction fetch/decode unit 10a-c to the functional units D1, M1, S1, and L1 over various sets of control lines which are not shown. Data are provided to/from the register file 20a from/to to load/store units D1 over a first set of busses 32a, to multiplier M1 over a second set of busses 34a, to ALU/shifter unit S1 over a third set of busses 36a and to ALU L1 over a fourth set of busses 38a. Data are provided to/from the memory 22 from/to the load/store units D1 via a fifth set of busses 40a. Note that the entire data path described above is duplicated with register file 20b and execution units D2, M2, S2, and L2. Instructions are fetched by fetch unit 10a from instruction memory 23 over a set of busses 41. Emulation unit 50 provides access to the internal operation of integrated circuit 1 which can be controlled by an external test system 51.

Note that the memory 22 and memory 23 are shown in Figure 20 to be a part of a microprocessor 1 integrated circuit, the extent of which is represented by the box 42. The memories 22-23 could just as well be external to the microprocessor 1 integrated circuit 42, or part of it could reside on the integrated circuit 42 and part of it be external to the integrated circuit 42. Also, an alternate number of execution units can be used.

When microprocessor 1 is incorporated in a data processing system, additional memory or peripherals may be connected to microprocessor 1, as illustrated in Figure 1. For example, Random Access Memory (RAM) 70, a Read Only Memory (ROM) 71 and a Disk 72 are shown connected via an external bus 73. Bus 73 is connected to the External Memory Interface (EMIF) which is part of functional block 61 within microprocessor 42. A Direct Memory Access (DMA) controller is also included within block 61. The DMA controller is generally used to move data between memory and peripherals within microprocessor 1 and memory and peripherals which are external to microprocessor 1.

Several example systems which can benefit from aspects of embodiments of the present invention are described in U.S. Patent 5,072,418, which was incorporated by reference herein, particularly with reference to Figures 2-18 of U.S. Patent 5,072,418. A microprocessor incorporating an aspect of an embodiment of the present invention to improve performance or reduce cost can be used to further improve the systems described in U.S. Patent 5,072,418. Such systems include, but are not limited to, industrial process controls, automotive vehicle systems, motor controls, robotic control systems, satellite telecommunication systems, echo canceling systems, modems, video imaging systems, speech recognition systems, vocoder-modem systems with encryption, and such.

A description of various architectural features of the microprocessor of Fig. 20 is provided in coassigned application serial number _____ (TI docket number T-25311). A description of a complete set of instructions for the microprocessor of Fig. 20 is also provided in coassigned application serial number _____ (TI docket number T-25311).

Figure 21 is a block diagram of the execution units and register files of the microprocessor of Fig. 20 and shows a more detailed view of the buses connecting the various functional blocks. In this figure, all data busses are 32 bits wide, unless otherwise noted. Bus 40a has an address bus DA1 which is driven by mux 200a. This allows an address generated by either load/store unit D1 or D2 to provide an address for loads or stores for register file 20a. Data Bus LD1 loads data from an address in memory 22 specified by address bus DA1 to a register in load unit D1. Unit D1 may manipulate the data provided prior to storing it in register file 20a. Likewise, data bus ST1 stores data from register file 20a to memory 22. Load/store unit D1 performs the following operations: 32-bit add, subtract, linear and circular address calculations. Load/store unit D2 operates similarly to unit D1, with the assistance of mux 200b for selecting an address.

ALU unit L1 performs the following types of operations: 32/40 bit arithmetic and compare operations; left most 1, 0, bit counting for 32 bits; normalization count for 32 and 40 bits; and logical operations. ALU L1 has input src1 for a 32 bit source operand and input src2 for a second 32 bit source operand. Input msb_src is an 8 bit value used to form 40 bit source operands. ALU L1 has an output dst for a 32 bit destination operands. Output msb_dst is an 8 bit value used to form 40 bit destination operands. Two 32 bit registers in register file 20a are concatenated to hold a 40 bit

operand. Mux 211 is connected to input src1 and allows a 32 bit operand to be obtained from register file 20a via bus 38a or from register file 20b via bus 210. Mux 212 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 38a or from register file 20b via bus 210. ALU unit L2 operates similarly to unit L1.

ALU/shifter unit S1 performs the following types of operations: 32 bit arithmetic operations; 32/40 bit shifts and 32 bit bit-field operations; 32 bit logical operations; branching; and constant generation. ALU S1 has input src1 for a 32 bit source operand and input src2 for a second 32 bit source operand. Input msb_src is an 8 bit value used to form 40 bit source operands. ALU S1 has an output dst for a 32 bit destination operands. Output msb_dst is an 8 bit value used to form 40 bit destination operands. Mux 213 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 36a or from register file 20b via bus 210. ALU unit S2 operates similarly to unit S1, but can additionally perform register transfers to/from the control register file 102.

Multiplier M1 performs 16×16 multiplies. Multiplier M1 has input src1 for a 32 bit source operand and input src2 for a 32 bit source operand. ALU S1 has an output dst for a 32 bit destination operands. Mux 214 is connected to input src2 and allows a 32 bit operand to be obtained from register file 20a via bus 34a or from register file 20b via bus 210. Multiplier M2 operates similarly to multiplier M1.

Figure 22A is a chart which illustrates the processing phases of an instruction execution pipeline in the microprocessor of Fig. 1. Each phase corresponds roughly to a clock cycle of a system clock. For example, if microprocessor 1 is being operated at 200 MHz, then each phase is nominally 5 Ns. However, in a phase where data is expected from a memory or peripheral, such as RAM 70, the pipeline will stall if the data is not ready when expected. When stalled, a given pipeline phase will exist for a number of system clock cycles.

In Figure 22A, the first phase of processing an instruction is to generate the program address in phase PG. This is done by loading a program fetch counter PFC which is located in control register file 102. During the second instruction processing phase PS, an address of an instruction fetch packet is sent to program memory 23 via a program address bus PADDR which is part of bus 41. The third phase PW is a wait phase to allow for access time in memory 23. During the fourth phase PR, a program fetch packet is available from program memory 23 via data bus PDATI which is part of bus 41. During the fifth processing phase DP, instruction parallelism is detected and instructions that can be executed are dispatched to the appropriate functional units. This aspect of pipeline operation will be described in more detail in later paragraphs. During the sixth processing phase DC, executable instructions are decoded and control signals are generated to control the various data paths and functional units.

Figure 22B is a chart which illustrates the execution phases of the instruction execution pipeline in the microprocessor of Fig 1. During the first execution phase E1, single cycle instructions, referred to as "ISC", and branch instructions, referred to as "BR", are completed. A designated execution unit performs the operations indicated in Fig. 22B as directed by control circuitry 100. During the second execution phase E2, the following types of instructions are completed by designated execution units under control of control circuitry 100: integer multiply (IMPY), program store instructions (STP), and data store instructions (STD). During the third execution phase E3, execution of load data instructions (LD) continues by latching data from the data memory system (DMS), as indicated. During execution phase E4, the data latched in E3 is transferred to a data input register DDATA_I in execution unit D1 or D2. During execution phase E5, the LD instruction is completed by manipulating the data in register DDATA_I and writing the manipulated data to a specified register in register file 20a or 20b.

Figure 23 is a timing diagram which illustrates timing details of processing an instruction fetch packet during the processing phases of Fig. 22A and execution of the execution packet during the execution phases of Fig. 22B. Note that a pipe stall is illustrated in phase PW due to a program memory ready signal PRDY being low in phase PS, and a second pipe stall in phase E3 due a data memory ready signal DRDY being low in phase E2.

Figure 24 is a block diagram showing instruction dispatching in the microprocessor of Fig. 20. In this embodiment, an instruction fetch packet contains eight instructions. Instruction fetch packet 1710 is dispatched and decoded to eight execution units as illustrated. Fetch packet 1720 contains a branch instruction 1725. Instruction fetch packet 1730 contains three instruction execute packets. The first execute packet contains two instructions, ZERO-SHL, which will begin processing in the first delay slot of branch instruction 1725. The second execute packet contains four instruction, ADD-SUB-STW-STW, which will begin processing in the second delay slot of branch instruction 1725. The third execute packet contains two instructions, ADDK-BR, which will begin processing in the third delay slot of branch instruction 1725.

Parallel Operations

Instructions are always fetched eight at a time. This constitutes a fetch packet. The basic format of a fetch packet is shown in Figure 25. The execution grouping of the fetch packet is specified by the p-bit, bit zero, of each instruction. Fetch packets are 8-word aligned.

The p bit controls the parallel execution of instructions. The p-bits are scanned from left to right (lower to higher address). If the p bit of instruction i is 1, then instruction i + 1 is to be executed in parallel with (in the same cycle as)

EP 0 855 648 A2

instruction i. If the p-bit of instruction i is 0, then instruction $i + 1$ is executed in the cycle after instruction i. All instructions executing in parallel constitute an execute packet. An execute packet can contain up to eight instructions. All instructions in an execute packet must use a unique functional unit.

An execute packet cannot cross an 8-word boundary. Therefore, the last p-bit in a fetch packet is always set to 0, and each fetch packet starts a new execute packet. As discussed with regard to Figures 4-19, there are three types of p-bit patterns for fetch packets. These three p-bit patterns result in the following execution sequences for the eight instructions: Fully serial, Fully parallel, Partially serial.

Example Parallel Code

The II characters signify that an instruction is to execute in parallel with the previous instruction. In the fetch packet of Figure 26, the code would be represented as this:

```
15          instruction A  
  
20          instruction B  
  
25          instruction C  
           II  instruction D  
           II  instruction E  
  
30          instruction F  
           II  instruction G  
           II  instruction H
```

Branching Into the Middle of an Execute Packet

If a branch into the middle of an execution packet occurs, all instructions at lower addresses are ignored. In the example in Figure 26, if a branch to the address containing instruction D occurs, then only D and E will execute. Even though instruction C is in the same execute packet, it is ignored. Instructions A and B are also ignored because they are in earlier execute packets. Resource Constraints

No two instructions within the same execute packet can use the same resources. Also, no two instructions can write to the same register during the same cycle. The following sections describe each of the resources an instruction can use.

Functional Units

Two instructions using the same functional unit cannot be issued in the same execute packet.
The following execute packet is invalid:

```
50          ADD  S1    A0, A1, A2      ; \ .S1 is used for  
           SHR .S1    A3, 15, A4      ; / both  
           instructions
```

The following execute packet is valid:

5 ADD L1 A0, A1, A2 ; \Two different
 functional
 SHR .S1 A3, 15, A4 ; / units are used

Cross Paths (1X and 2X)

10 One unit (either a .S, .L, or .M) per data path, per execute packet, can read a source operand from its opposite register file via the cross paths (1X and 2X). For example, .S1 can read both operands from the A register file, or one operand from the B register file using the 1X cross path. This is denoted by an X following the unit name.

15 Two instructions using the same X cross path between register files cannot be issued in the same execute packet since there is only one path from A to B and one path from B to A.

15 The following execute packet is invalid:

20 ADD.L1X A0,B1,A1 ; \ 1X cross path is used
 MPY.M1X A4,B4,A5 ; / for both instructions

The following execute packet is valid:

25 ADD.L1X A0,B1,A1 ; \ Instructions use the 1x
 and
 MPY.M2X A4,B4,132 ; / 2x cross paths

30 The operand will come from a register file opposite of the destination if the x bit in the instruction field is set.

Load and Store Path

35 Loads and stores can use an address pointer from one register file while loading to or storing from the other register file. Two loads and/or stores using an address pointer from the same register file cannot be issued in the same execute packet.

The following execute packet is invalid:

40 LDW D1 *A0,A1; \ Address registers from the
 same
 || LDW.D1 *A2,B2; / register file

45

The following execute packet is valid:

50 LDW D1 *A0,A1; \ Address registers from
 different
 || LDW.D2 *B0,B2 ; / register files

55

Two loads and/or stores loading to and/or storing from the same register file cannot be issued in the same execute packet.

The following execute packet is invalid:

LDW D1 *A4,A5; \ Loading to and storing from
 the
 5 || STW.D2 A6,*B4; / same register file

The following execute packet is valid:

10 LDW D1 *A4,B5; \ Loading to, and storing
 from
 15 || STW.D2 A6,*B4; / different register files

Long Paths

20 Only one long result may be written per cycle on each side of the register file. Because the S and .L units share a read register port for long source operands and a write register port for long results, only one may be issued per side in an execute packet.

The following execute packet is invalid:

25 ADD.L1 A5:A4,A1,A3 :A2; \ Two long writes
 SHL.S1 A8, A9, A7 :A6 ; / on A register
 file

The following execute packet is valid:

30 ADD.L1 A5:A4,A1,A3 :A2; \ One long write for
 35 SHL.S2 B8, B9, B7 :B6 ; / each register
 file

40 Because the .L and .S units share their long read port with the store port, operations that read a long value cannot be issued on the .L and/or .S units in the same execute packet as a store.

The following execute packet is invalid:

45 ADD.L1 A5:A4,A1,A3 :A2; \ Long read operation
 and
 || STW.D1 A8, *A9 ; / a store

50 The following execute packet is valid:

55 ADD.L1 A4, A1, A3 :A2 ; \ No long read with
 || STW.D1 AB, *A9 ; / with the store

Register Reads

More than four reads of the same register cannot occur on the same cycle. Conditional registers are not included in this count.

- 5 The following code sequence is invalid:

```

      MPY      .M1  A1,A1,A4 ; five reads of
10     register A1
      ||  ADD      .L1  A1,A1,A5
      ||  SUB      .D1  A1,A2,A3
    
```

- 15 Whereas this code sequence is valid:

```

      MPY      .M1  A1,A1,A4 ; only four reads of
20     A1
      |  [A1] ADD  .L1  A0,A1,A5
      |  SUB      .D1  A1,A2,A3
    
```

25 Register Writes

Multiple writes to the same register on the same cycle can occur if instructions with different latencies writing to the same register are issued on different cycles. For example, an MPY issued on cycle i followed by an ADD on cycle i+1 cannot write to the same register since both instructions will write a result on cycle i+1. Therefore, the following code sequence is invalid:

MPY	.M1	A0,A1,A2
ADD	.L1	A4,A5,A2

35 Detectability of Write Conflicts

The following sequence of execute packets shows different multiple write conflicts. For example, the ADD and SUB in execute packet L1 write to the same register. This conflict is easily detectable.

40

45

50

55

```

L1:      ADD.L2    B5,B6,B7 ; detectable,
conflict
      ||      SUB.S2    B8;B9,B7
      L2:      MPY.M2    B0,B1,B2 ; \ not
detectable
      L3      ADD.L2    B3,B4,B2 ; /
      L4:  [ !B0 ] ADD.L2    B5,B6,B7 ; detectable, no
conflict
      ||  [B0]      SUB.S2    B8, B9 B7
      L5:  [ !B1 ]      ADD.L2    B5,B6,B7 ; \ not
detectable
      ||  [B0]      SUB.S2    B8,B9,B7 ; /

```

The MPY in packet L2 and the ADD in packet L3 might both write to B2 simultaneously; however, if a branch instruction causes the execute packet after L2 to be something other than L3, this would not be a conflict. Thus, the potential conflict in L2 and L3 might not be detected by the assembler. The instructions in L4 do not constitute a write conflict because they are mutually exclusive. In contrast, because it is not obvious that the instructions in L5 are mutually exclusive, the assembler cannot determine a conflict. If the pipeline does receive commands to perform multiple writes to the same register, the result is undefined.

Although exemplary embodiments of the present invention are described above, this does not limit the scope of the invention, which can be practiced in a variety of embodiments.

The scope of the present disclosure includes any novel feature or combination of features disclosed therein either explicitly or implicitly or any generalisation thereof irrespective of whether or not it relates to the claimed invention or mitigates any or all of the problems addressed by the present invention. The applicant hereby gives notice that new claims may be formulated to such features during the prosecution of this application or of any such further application derived therefrom. In particular, with reference to the appended claims, features from dependent claims may be combined with those of the independent claims in any appropriate manner and not merely in specific combinations enumerated in the claims.

40 Claims

1. A data processing system, comprising:

45 data processing circuitry having circuitry for producing a set of instructions which include respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions; and
said data processing circuitry including program execution circuitry connected to said producing circuitry for receiving the set of instructions and selectively responsive to said instruction portions for executing simultaneously a plurality of said instructions indicated by said instruction portions.

50 2. A method of processing program instructions in a data processing system, comprising the steps of:

providing a set of instructions with respective instruction portions for indicating whether the respective instructions can be executed simultaneously with another of the instructions; and
determining from said instruction portions whether a plurality of said instructions can be executed simultaneously.

55 3. A method of compiling a program for execution by a data processing system, comprising the steps of:

EP 0 855 648 A2

determining whether a first program instruction can be executed simultaneously with a second program instruction that immediately sequentially follows the first program instruction in program; and providing the first instruction with an instruction portion that indicates whether the first instruction can be executed simultaneously with the second instruction.

5

10

15

20

25

30

35

40

45

50

55

FIG. 1

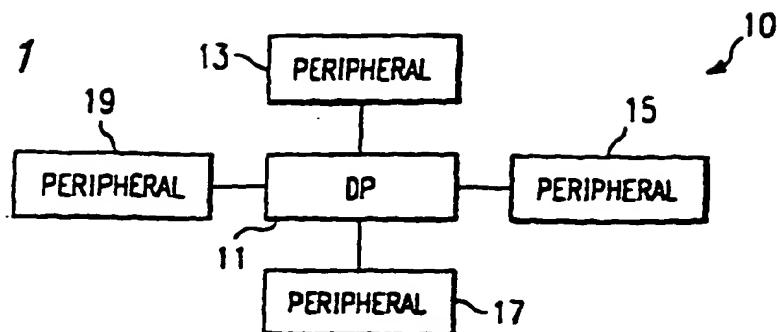


FIG. 2

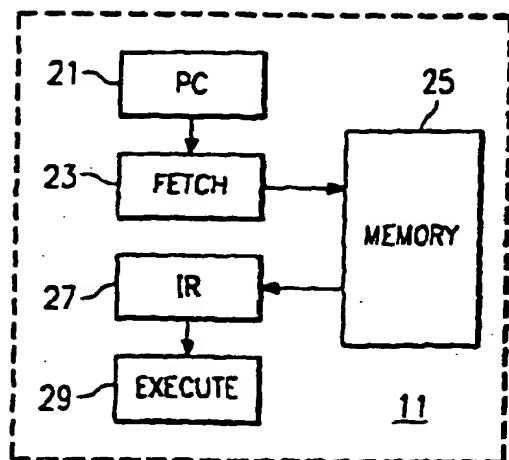


FIG. 5

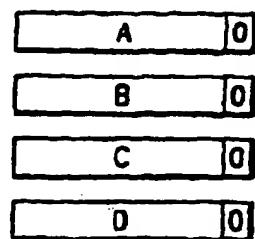


FIG. 3

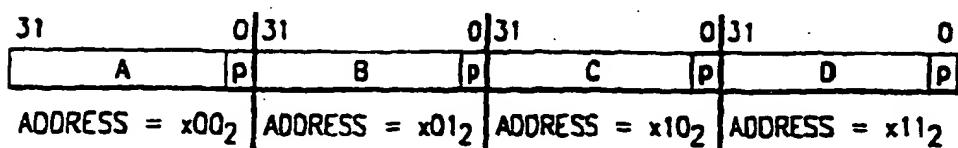


FIG. 4

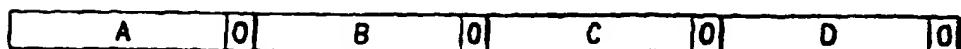


FIG. 6



FIG. 7



FIG. 8



FIG. 9

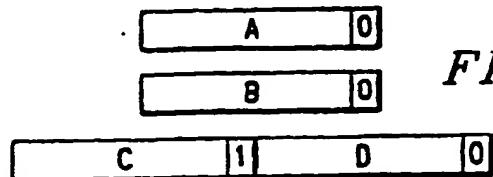


FIG. 10



FIG. 11

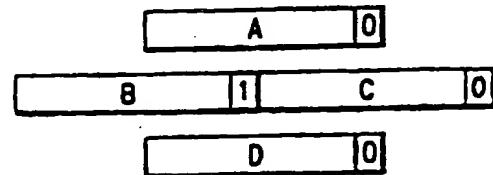


FIG. 12



FIG. 13

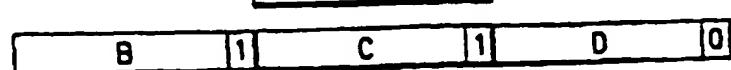


FIG. 14

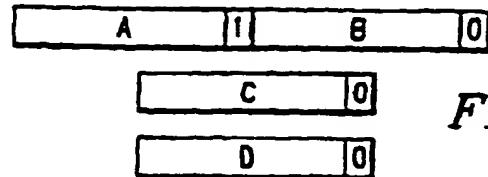


FIG. 15

FIG. 16

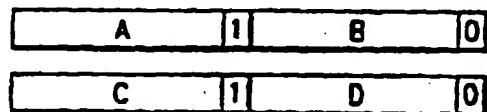
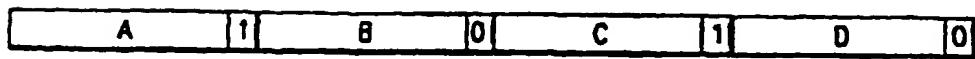


FIG. 17

FIG. 18

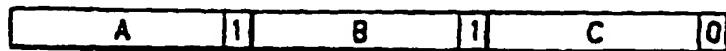
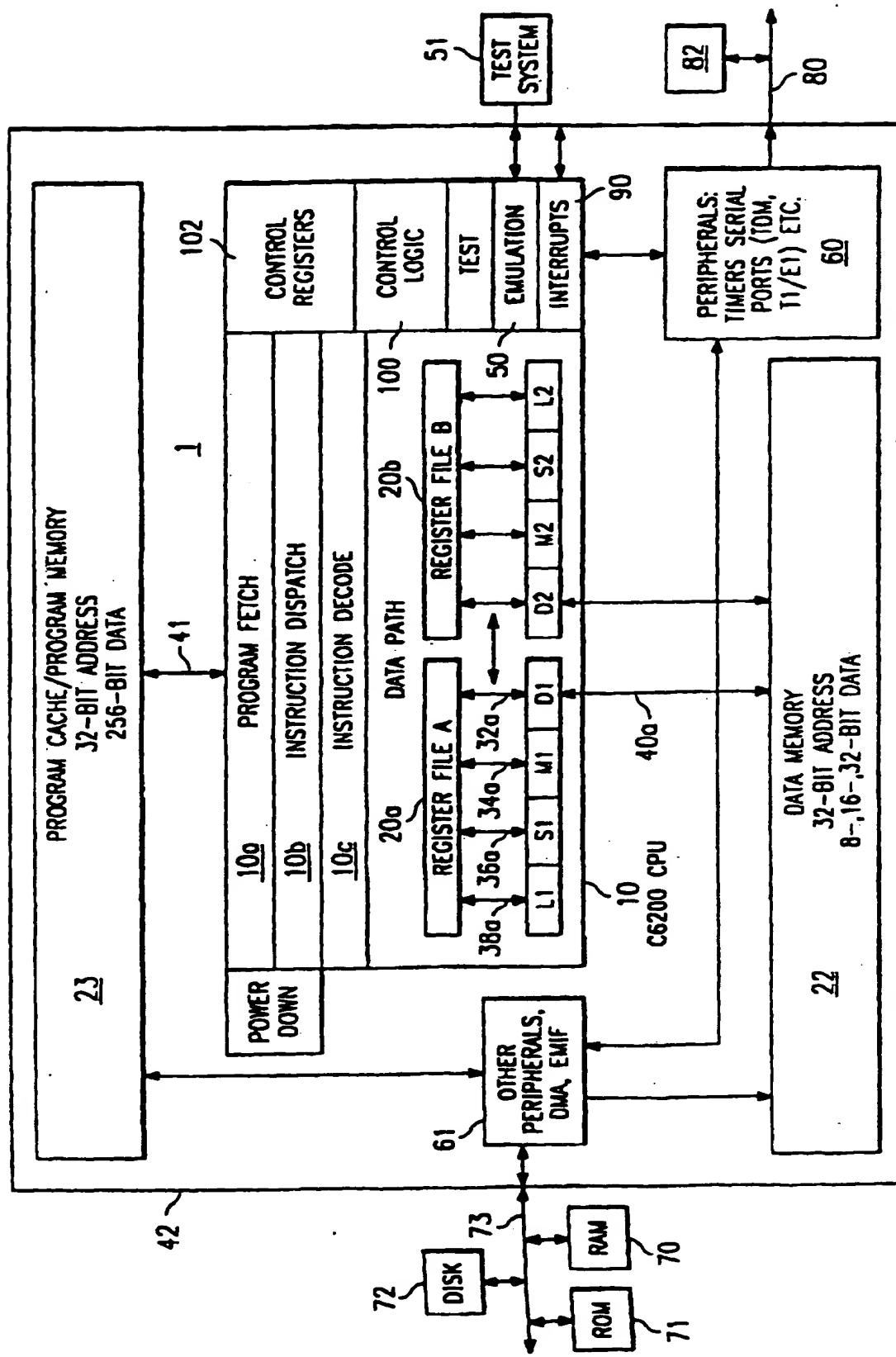


FIG. 19

FIG. 20



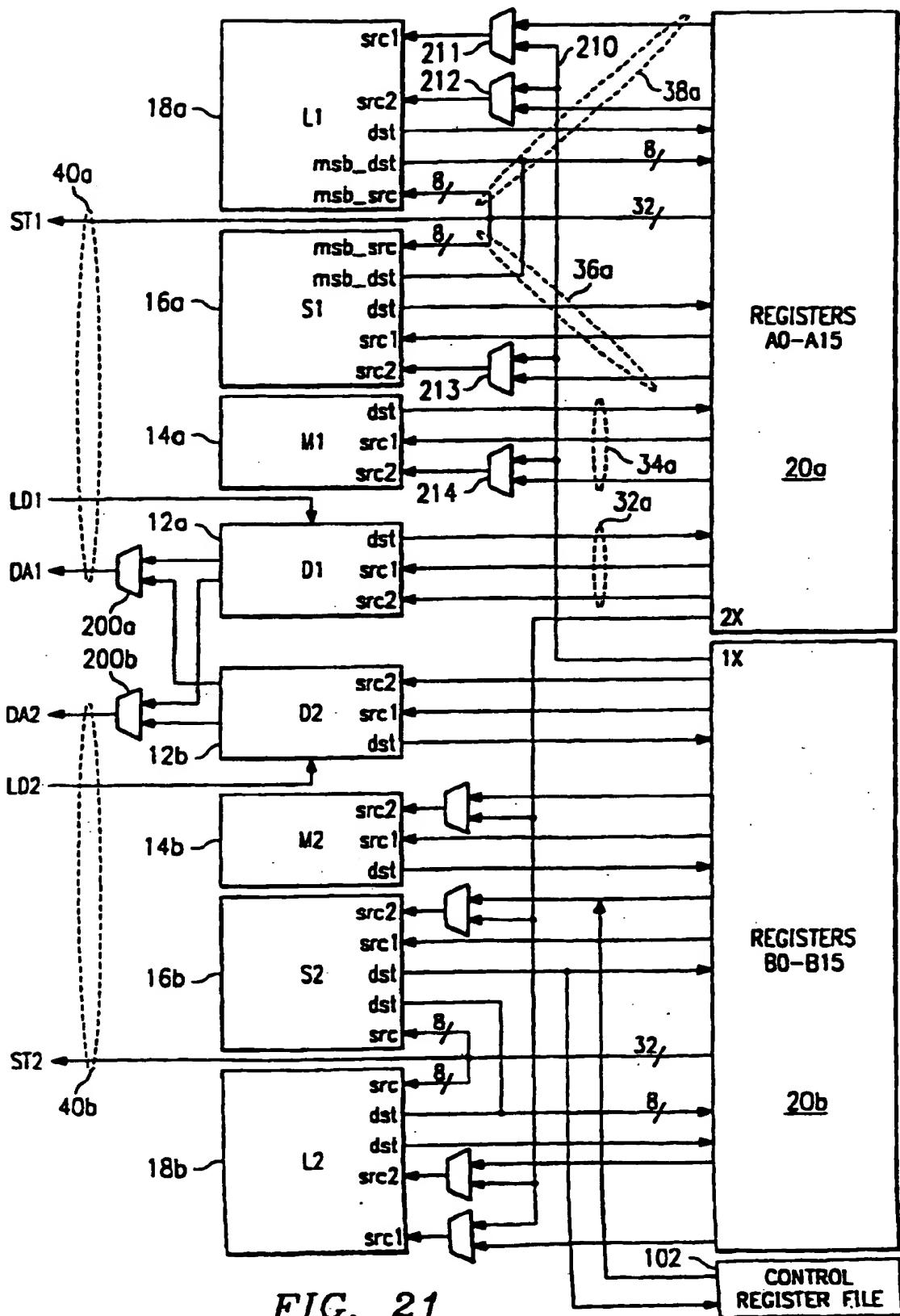


FIG. 21

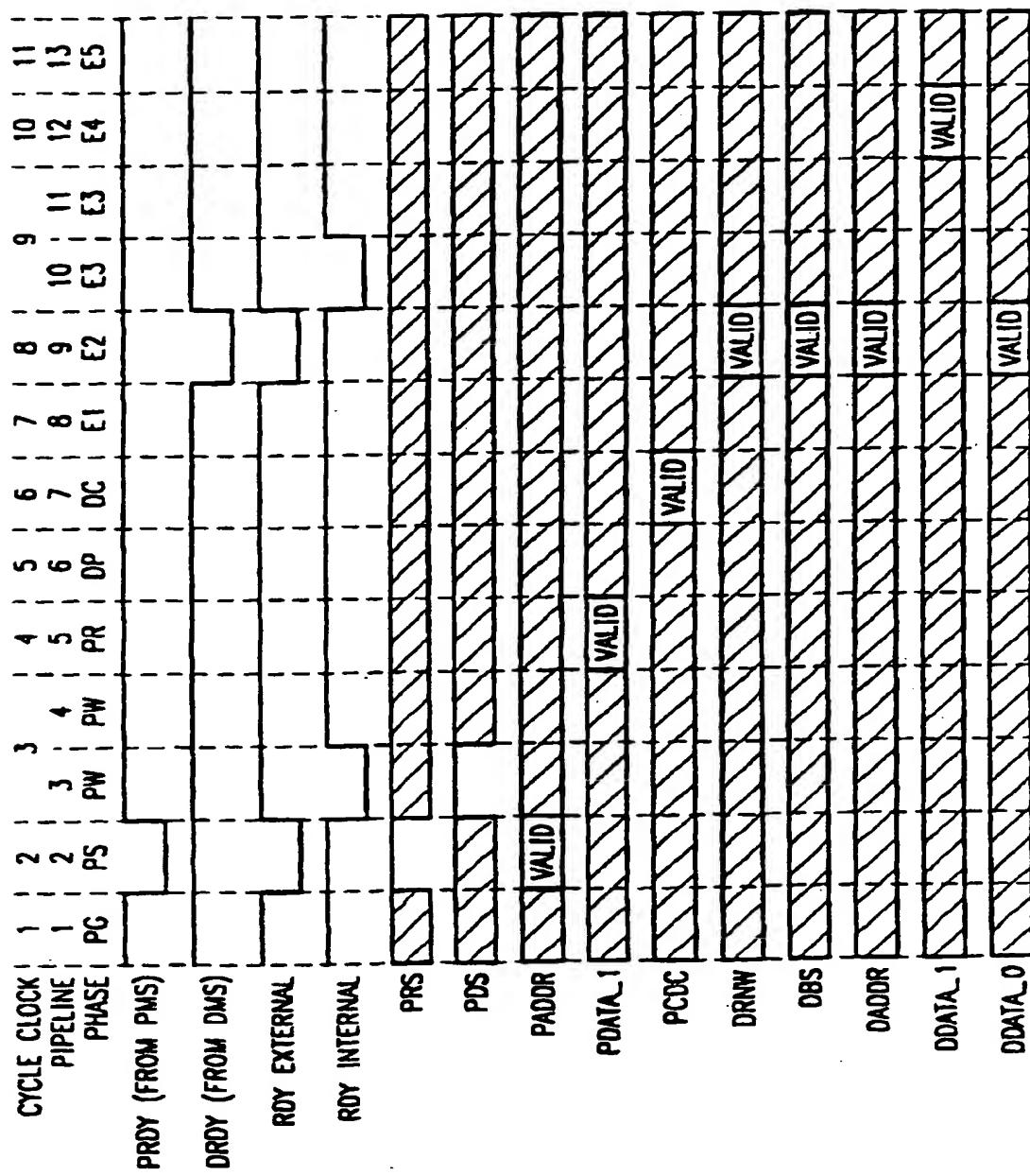
FIG. 22A

Pipeline Phase.	Symbol	During this Entire Phase	By the End of This Phase
Program Address Generate	PC		<ul style="list-style-type: none"> □ Except when a program store is in E1, PFC and PADDR and latched to their next values: either the next 8-word boundary after the current PFC or a branch target address. □ When a program store is in E1, the PFC remains unchanged and PADDR takes the value from the program store instruction.
Program Address Send	PS	<ul style="list-style-type: none"> □ PFC valid on PADDR. □ If a fetch packet fetch needed, PRS set active. 	<ul style="list-style-type: none"> □ If PRS active, PMS latches PADDR.
Program Wait	PW	<ul style="list-style-type: none"> □ If the fetch packet indicated by the last PRS is necessary, PDS is asserted. 	<ul style="list-style-type: none"> □ If PDS is asserted, PWS latches the last requested fetch packet to be driven during PR.
Program Data Receive	PR	<ul style="list-style-type: none"> □ If PDS active during the associated PW, PMS drives requested fetch packet onto PDATA_1. Otherwise, this fetch maintains state. 	<ul style="list-style-type: none"> □ If it can be determined that the fetch packet requested by an active PDS in PW is needed, CPU latches this fetch packet on PDATA_1. Otherwise, this latches maintains state.
Dispatch	DP		<ul style="list-style-type: none"> □ Parallelism of the fetch packet detected. □ Instructions in the next required execute packet in the fetch packet dispatched to the appropriate functional units. □ Detection of the software breakpoint code in the CREG field detected. □ Multi-Cycle NOP, NOP, SWI, and IDLE decoded because they are not passed to individual functional units. □ PC of the execute packet in DP latched at the end of the cycle as PCDC. □ Control signals for actions during execute decoded.
Decode	DC	□ PCDC valid.	

FIG. 22B

Pipeline Phase	Symbol	During this Phase	By the End of This Phase	CPU has Completed Types
		<input type="checkbox"/> All Types: Registers read.	<ul style="list-style-type: none"> <input type="checkbox"/> Type ISS: Results written to the register file. <input type="checkbox"/> Type ISS: Status written to the control register file. <input type="checkbox"/> Type LD and ST: Address modifications written to the register file. <input type="checkbox"/> Type LD and ST: DADDR, DBS, and DRNW latched for next phase. <input type="checkbox"/> Type ST: DDATA_0 latched for next phase. <input type="checkbox"/> Type BRANCH: PFC and PADDR latches the target address. <input type="checkbox"/> Type STP: PADDR latches the destination address if no fetch is pending. PWS latched as active for next phase. <input type="checkbox"/> Types LD, ST, STP, IMPY: Intermediate states latched. 	<input type="checkbox"/> ISC. <input type="checkbox"/> BR.
Execute 1	E1			
		<input type="checkbox"/> Types LD and ST: DBS active and DADDR valid.	<input type="checkbox"/> Type IMPY: Results written to the register file. ³	<input type="checkbox"/> IMPY.
		<input type="checkbox"/> Type LD: DRNW active.	<input type="checkbox"/> Type IMPY: Status written to the control register file.	<input type="checkbox"/> STD. <input type="checkbox"/> STP.
Execute 2	E2	<input type="checkbox"/> Type ST: DRNW inactive and DDATA_0 valid.	<ul style="list-style-type: none"> <input type="checkbox"/> Types LD and ST: DMS latches ADDR. <input type="checkbox"/> Type ST: DMS latches DDATA_0. <input type="checkbox"/> Type STP: PWS latches PDATA_0. <input type="checkbox"/> Type LD: Intermediate states latched. 	
			<ul style="list-style-type: none"> <input type="checkbox"/> Type LD: DMS latches data requested during the last cycle. <input type="checkbox"/> Types LD: Intermediate states latched. 	
Execute 3	E3		<input type="checkbox"/> Type LD: DMS drives on DDATA_1 data requested during associated E2.	<input type="checkbox"/> Type LD: DDATA_1 latched.
Execute 4	E4		<input type="checkbox"/> Types LD: Intermediate states latched.	<input type="checkbox"/> Types LD: Intermediate states latched.
Execute 5	E5		<input type="checkbox"/> Type LD: DDATA_1 right-aligned, either sign-extended or zero-filled, and written to the register file.	<input type="checkbox"/> LD.

FIG. 23



1723 ~ PROGRAM MEMORY/256-BIT DATA

FIG. 24

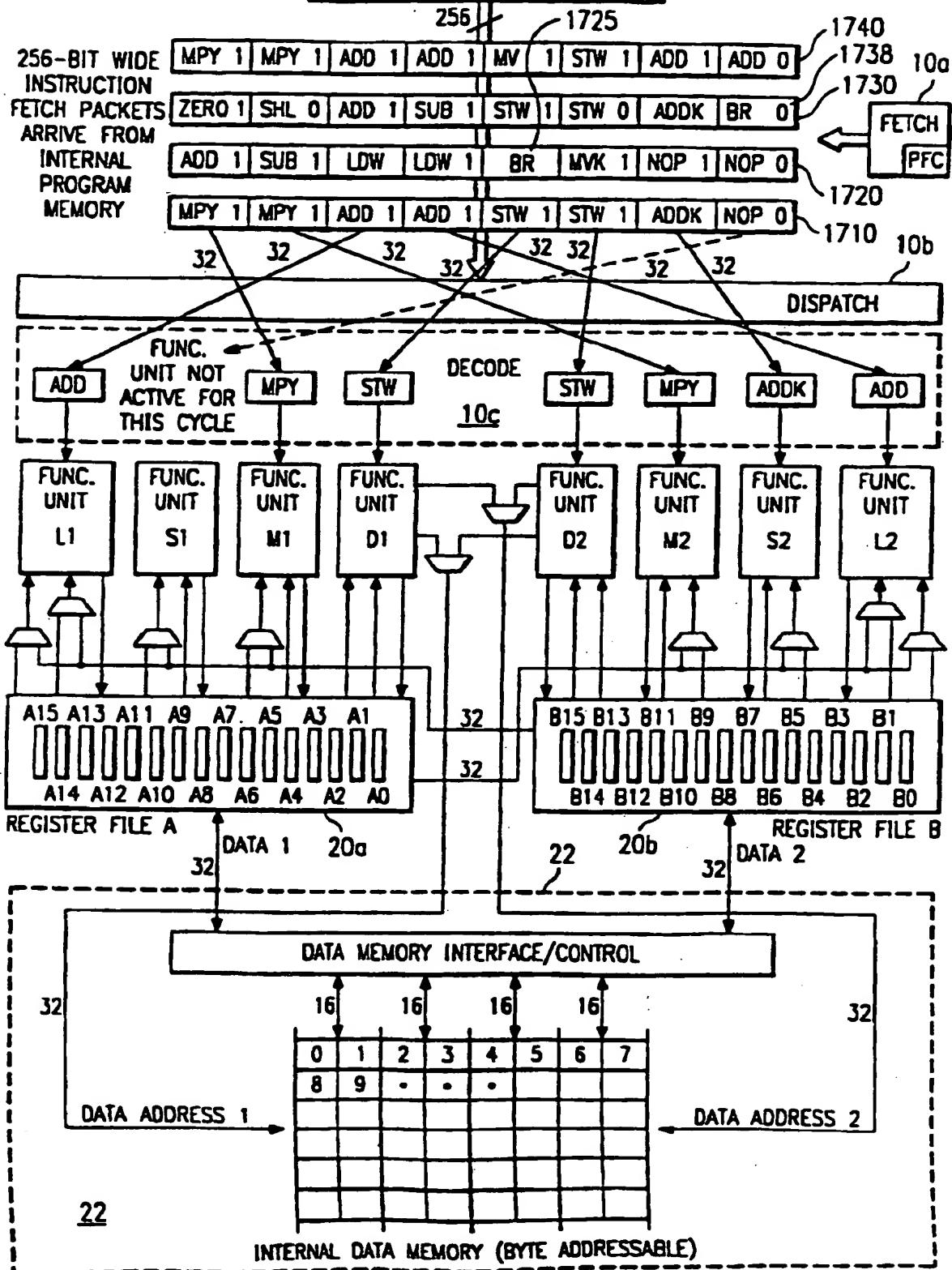


FIG. 25

31	0 31	0 31	0 31	0 31	0 31	0 31	0 31	0 31	0 31	0
	• p	• p	• p	• p	• p	• p	• p	• p	• p	• p
Instruction A		Instruction B		Instruction C		Instruction D		Instruction E		Instruction F
LSBs of the byte address	00000 ₂	00100 ₂	01000 ₂	01100 ₂	10000 ₂	10100 ₂	11000 ₂	11100 ₂		Instruction G

This p-bit pattern:

31	0 31	0 31	0 31	0 31	0 31	0 31	0 31	0 31	0 31	0
	• 0	• 0	• 1	• 1	• 0	• 1	• 1	• 1	• 0	• 0
Instruction A		Instruction B		Instruction C		Instruction D		Instruction E		Instruction F

results in this execution sequence:

Cycle	Instructions
1	A
2	B
3	C D E
4	F G H

Note: Instructions C, D, and E do not use any of the same functional units, cross paths, or other data path resources. This is also true for instructions F, G, and H.

FIG. 26



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 30 0494

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	EP 0 455 966 A (IBM) 13 November 1991 (1991-11-13) * column 2, line 47 - column 3, line 20 * * column 6, line 12 - line 30 *	1-3	G06F9/38
X	VASSILIADIS S ET AL: "ON THE ATTRIBUTES OF THE SCISM ORGANIZATION" COMPUTER ARCHITECTURE NEWS, vol. 20, no. 4, 1 September 1992 (1992-09-01), pages 44-53, XP000298584 ISSN: 0163-5964 * page 45, right-hand column, line 32 - line 50 * * page 46, left-hand column, line 30 - line 33 *	1-3	
X	US 5 504 932 A (JEREMIAH THOMAS L ET AL) 2 April 1996 (1996-04-02) * column 3, line 24 - line 64 *	1-3	
X	PATENT ABSTRACTS OF JAPAN vol. 11, no. 93 (P-559), 24 March 1987 (1987-03-24) & JP 61 245239 A (TOSHIBA CORPORATION), 31 October 1986 (1986-10-31) * abstract *	1-3	TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F
X	US 5 481 751 A (TALMUDI RAN ET AL) 2 January 1996 (1996-01-02) * column 3, line 20 - line 37 *	1-3	
X	EP 0 667 571 A (HEWLETT PACKARD CO) 16 August 1995 (1995-08-16) * page 3, line 1 - line 17 *	1-3	
		-/-	
The present search report has been drawn up for all claims			
Place of search	Date of completion of the search	Examiner	
MUNICH	25 October 1999	Thibaudeau, J	
CATEGORY OF CITED DOCUMENTS		T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document	
X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document			



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 30 0494

DOCUMENTS CONSIDERED TO BE RELEVANT			CLASSIFICATION OF THE APPLICATION (Int.Cl.)
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	
X	US 5 337 415 A (DELAND ERIC R ET AL) 9 August 1994 (1994-08-09) * column 3, line 61 - column 4, line 27 *	1-3	
			TECHNICAL FIELDS SEARCHED (Int.Cl.)
The present search report has been drawn up for all claims			
Place of search	Date of completion of the search		Examiner
MUNICH	25 October 1999		Thibaudeau, J
CATEGORY OF CITED DOCUMENTS X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document			
T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document			

**ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.**

EP 98 30 0494

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

25-10-1999

Patent document cited in search report		Publication date		Patent family member(s)	Publication date
EP 0455966	A	13-11-1991	US	5214763 A	25-05-1993
			AT	131637 T	15-12-1995
			CA	2040304 C	11-04-1995
			CS	9100933 A	13-05-1992
			DE	69115344 D	25-01-1996
			DE	69115344 T	20-06-1996
			HU	214423 B	30-03-1998
			JP	2039125 C	28-03-1996
			JP	4230528 A	19-08-1992
			JP	7076924 B	16-08-1995
			JP	2046578 C	25-04-1996
			JP	6083623 A	25-03-1994
			JP	7078737 B	23-08-1995
			PL	165491 B	30-12-1994
			US	5465377 A	07-11-1995
			US	5475853 A	12-12-1995
			US	5295249 A	15-03-1994
US 5504932	A	02-04-1996	US	5303356 A	12-04-1994
			US	5197135 A	23-03-1993
			US	5295249 A	15-03-1994
			CA	2053941 A	30-09-1992
			US	5446850 A	29-08-1995
			US	5701430 A	23-12-1997
			AT	146611 T	15-01-1997
			CA	2039640 A	05-11-1991
			CA	2040637 A, C	05-11-1991
			CS	9100936 A	17-12-1991
			DE	69123629 D	30-01-1997
			DE	69123629 T	12-06-1997
			EP	0454985 A	06-11-1991
			EP	0481031 A	22-04-1992
			EP	0545927 A	16-06-1993
			EP	0825529 A	25-02-1998
			JP	2500082 B	29-05-1996
			JP	4229326 A	18-08-1992
			JP	6080489 B	12-10-1994
			JP	4505823 T	08-10-1992
			JP	6079273 B	05-10-1994
			JP	4506878 T	26-11-1992
			PL	165524 B	31-01-1995
			PL	166513 B	31-05-1995
			WO	9117496 A	14-11-1991
			WO	9117495 A	14-11-1991
			US	5465377 A	07-11-1995
			US	5448746 A	05-09-1995

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

**ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.**

EP 98 30 0494

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on. The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

25-10-1999

Patent document cited in search report	Publication date		Patent family member(s)	Publication date
US 5504932 A			US 5502826 A US 5732234 A EP 0463299 A JP 2046580 C JP 4233034 A JP 7082438 B EP 0496928 A JP 2046578 C JP 6083623 A JP 7078737 B CA 2037708 A,C CS 9100935 A DE 69122294 D DE 69122294 T EP 0454984 A JP 7073036 A JP 8007681 B US 5500942 A	26-03-1994 24-03-1998 02-01-1992 25-04-1996 21-08-1992 06-09-1995 05-08-1992 25-04-1996 25-03-1994 23-08-1995 05-11-1991 17-12-1991 31-10-1996 10-04-1997 06-11-1991 17-03-1995 29-01-1996 19-03-1996
JP 61245239 A	31-10-1986		NONE	
US 5481751 A	02-01-1996		US 5669011 A DE 69130588 D DE 69130588 T EP 0459232 A JP 5053795 A	16-09-1997 21-01-1999 27-05-1999 04-12-1991 05-03-1993
EP 0667571 A	16-08-1995		DE 69510966 D JP 7253887 A	02-09-1999 03-10-1995
US 5337415 A	09-08-1994		JP 6274652 A	30-09-1994